第 42 卷　第 1 期　　天 文 学 进 展　　Vol. 42,　No. 1
2024 年 3 月　　PROGRESS IN ASTRONOMY　　Mar., 2024

doi: 10.3969/j.issn.1000-8349.2024.01.06

基于 Spark 分布式框架的海量星表数据时序重构方法研究

赵　青[1]，权文利[1]，陈亚瑞[1,2]，崔辰州[3]，樊东卫[3]

(1. 天津科技大学　人工智能学院，天津　300457；　2. 数字化学习技术集成与应用教育部工程研究中心，北京　100039；　3. 中国科学院　国家天文台，北京　100101)

摘要： 时序重构是时域天文学中的一个重要数据处理步骤，也是拟合光变曲线、开展时域分析研究的基础。Hadoop、Spark 这类 MapReduce 分布式模型在执行过程中分布式集群节点间的任务比较独立，需要跨节点的数据传输量较少。提出了非阻塞异步执行流程，每个分布式进程完全针对独立天区的数据进行连续处理，而分块边缘的新增天体导致的其他节点的新增证认任务延时批量追加，并且会根据各进程间的进度不同确定追加方式，保证证认计算没有遗漏，从而在提高并发效率的同时保证算法的精度。此外，对两表间的不同 Join 策略从理论和实验两个角度进行了研究并提出了免 Join 策略。最后通过基于 Spark 分布式框架的高效时序重构系统的设计完成了以上研究的验证。实验表明，与以往研究结果相比，该时序重构算法效率提升明显，为时域天文学中的天文时序数据分析的开展打下了良好的基础。

关　键　词： 时域天文学；交叉证认计算；时序重构；分布式计算；Spark

中图分类号： P129　　　　**文献标识码：** A

1　引　言

在时域天文学中，如何对相同望远镜的连续多次观测记录进行交叉证认以确定每一天体在不同观测记录间的对应关系，从而生成时序数据，是开展时域天文学研究的基础。这一过程涉及的数据量之大与速度要求之高往往构成矛盾。以地基广角相机阵 (Ground Wide Angle Cameras, GWAC)[1] 为例，这一数据处理过程需要在一个曝光周期内完成。因此，设计开发面向大视场时域巡天观测的高效时序数据生成方法具有重要意义。

收稿日期： 2023-06-13；　**修回日期：** 2023-11-07

资助项目： 国家自然科学基金 (11803022, 12273077)；国家重点研发计划 (2022YFF0711500)；数字化学习技术集成与应用教育部工程研究中心创新基金 (1221004)

通讯作者： 陈亚瑞，yrchen@tust.edu.cn

近年来，时域观测技术的进步为时域天文学研究提供了极其丰富的数据资源，例如盖亚 (Gaia) DR3 发布了 34 个月收集的 15.9 亿个源和 4.7 亿个天体的天体物理参数[2]。再如 GWAC，每 15 s 产生 1.1 G 数据，每个观测夜入库的仅星表数据可以高达 2 TB[3]；大型综合巡天望远镜 (Large Synoptic Survey Telescope, LSST)[4]每晚产生的数据量达到 15 TB，10 年期的运行预计收集到 60 PB 的数据。面对如此庞大的积累数据和每天都在高速产生的新归档数据，传统的交叉证认算法很难同时满足精度和效率要求。

首先是数据分块后出现的边界漏源问题对精度和效率的影响。当前的主要解决方法包括：Zhao 等人[5]在每个星表中增加了额外的边缘数据来解决该问题，但是实现复杂且效率低下。Du 等人[6]和 Yu 等人[7]采用了基于 HTM 和 HEALPix 两种索引的混合证认计算方法，虽减少了跨界点的数据传输量，但存储量、计算量明显增加，且仍存在一定比例的漏源，精度有所损失。因此边缘漏源问题有必要继续优化。

另一方面，时序证认需要在连续观测的多星表间进行交叉证认计算，因此计算量比传统多波段交叉证认更高，需要在并行计算框架方面进一步展开研究。Du 等人[6]提出了一种基于多线程的证认算法来加速计算的过程；Zhao 等人[5]提出的多核并行交叉证认算法，能够对 1 亿多记录和 4.7 亿多记录的星表进行交叉证认。然而这些方法由于未在大规模分布式环境下执行，对于多星表间的时序重构需求，仍然存在一定的性能限制。在同源星表的时序研究中，Yu 等人[7]在 MPI 并行环境下实现了同源星表的时序重构，最多实现了在 6 个进程下的并行执行。尽管提高了证认效率，但该并行环境不易扩展规模，对于进一步扩大的数据量难以应对。徐丹滢等人[8]提出了一种基于 MongoDB 的快速时序重构算法，该方法对时序重构问题的非关系型数据选择方面有一定参考价值，但在大规模分布式框架方面结合不足。因此，针对大规模同源星表的时序重构，需要引入新型并行框架，在保证精度的基础上进一步提高效率。

以 Hadoop、Spark 等 MapReduce 生态系统为代表的云计算逐渐成为大数据处理难题的重要解决方案。其中，Spark 作为基于内存计算的通用分布式框架，更具有效率优势。近年来已在天文信息处理方面有所应用，如 Brahem 等人[9]利用 Spark 平台的高效处理数据特性，提出了一种天文数据查询处理方法。Zečeviě 等人[10]基于 Spark 提出了一种可扩展天文分析框架。刘莹等人[11]结合 MapReduce/Spark 提出一种基于无监督聚类的脉冲星候选体筛选方案。可见，采用分布式计算技术已成为海量数据下进行天文研究工作的一种发展趋势。

但是，直接应用 Spark 在这一问题上并不适合。因为 Hadoop、Spark 这类 MapReduce 分布式模型需要执行过程中多进程任务间较为独立，否则频繁的节点间通信会成为这一模型应用的效率瓶颈。但因为交叉证认中经典问题——边界漏源问题——的存在，边缘新增数据的传输无法避免，这将严重降低这一模型实际应用中的加速比。因此，我们设计了非阻塞异步执行流程，每个分布式进程完全针对独立天区的数据进行非阻塞处理，新增星体导致的旁节点新增证认任务延迟批量追加，在保证证认计算没有遗漏的前提下显著提高并发效率。

此外，在将交叉证认计算转化为两表间的 Join 操作时，采用何种优化策略也是效率的关键。综合以上问题，本文设计了适合海量星表的分布式时序重构算法，并在以下三方面进行重点优化：(1) 针对边界漏源问题，设计了适合于 Spark 架构的非阻塞异步执行流程，在

保证精度的前提下减少了数据传输量，有效提高了算法效率；(2) 将时序证认转化为大表、小表间的 Join 运算，对比研究了两种不同的 Join 实现策略和免 Join 运算策略，给出了不同算法的适用情况；(3) 通过基于 Spark 分布式框架的时序证认系统的设计和实现对本文研究内容进行了验证。

2 异步非阻塞多进程执行模式的设计和数据的分布式更新同步

星表在分布式数据划分后会出现边界漏源问题，即由于误差的存在，某些处于划分边缘的星体在另一次观测中的对应体有可能会跨过边界，出现在另一块中。因此边缘数据需要额外的处理才能提高匹配精度。又由于在时序重构中，交叉证认出现在连续观测星表间，新增的星体如果处于分块边缘，就需要跨节点的同步更新。如果分布式架构中的多节点采用阻塞模式运行，可以保证新增星体的全局更新，但这意味着每次观测记录的全部星体计算完成后才能开启下一次观测数据的计算，这会制约整个分布式框架效率优势的发挥，并导致计算资源的浪费。

为了解决上述问题，本文提出的异步非阻塞多进程执行模式可以显著提高效率和并发处理能力。所谓非阻塞模式，这里指当处于边缘的新增星体更新到邻接块后，邻接块并不会停止当前的批处理计算而对新增数据进行处理，而是继续星表的批处理证认计算，各个进程在星表的处理进度上彼此独立，每个进程都可以高效率地逐个处理后续星表的该块内数据。而各个进程因为处理进度不同步造成的新增数据的证认遗漏将在批量证认计算结束后集中追加。这种非阻塞多进程模式可以最大限度地利用各个节点的计算资源，实现高效率并行处理。本文将证认时间轴上的最早星表作为参考星表，后续观测记录中的每个星体将与同一个分区的参考星表中的星体进行两两球面距离计算，计算公式为：

$$d = \arccos[\sin\delta_1 \sin\delta_2 + \cos\delta_1 \cos\delta_2 \cos(\alpha_1 - \alpha_2)] \quad , \tag{1}$$

其中，$(\alpha_1,\ \delta_1)$ 为天体 A 的坐标，$(\alpha_2,\ \delta_2)$ 为天体 B 的坐标。当两个天体的角距离非常小时，角距离公式就可以近似为：

$$d = \sqrt{[(\alpha_1 - \alpha_2)\cos\delta]^2 + (\delta_1 - \delta_2)^2} \quad , \tag{2}$$

其中，

$$\delta = (\delta_1 + \delta_2)/2 \quad . \tag{3}$$

由于存在一定的误差半径，假设两个星表的误差半径分别为 r_1 和 r_2，如果两个天体满足以下关系就可以判定其为同一天体：

$$d \leqslant |r_1| + |r_2| \quad , \tag{4}$$

或者，

$$d \leqslant 3\sqrt{r_1^2 + r_2^2} \quad . \tag{5}$$

由于不同观测星表都与参考星表进行证认计算，因此只需要对参考星表进行边缘副本冗余存储，减少了副本存储数据量。证认过程中出现的少量新增天体会追加在参考星表，如若处于分块边缘，还需要对多个计算节点的参考星表同时进行更新。

由于设计了非阻塞多进程计算模式，每个分布式进程完全针对独立天区的数据进行连续处理，而处于分块边缘的新增星体在跨节点同步后，如果出现了进度超前而遗漏了证认计算的情况，将在后期进行批量弥补。由此可见，系统的并发性更强，计算资源利用率更高，更适合于海量星表的时序重构计算。

具体的更新流程示意图如图 1 所示。由于证认计算任务是异步执行，所以当前块与相邻块证认进度并不同步，具体新增星体的更新同步问题的解决方法为：

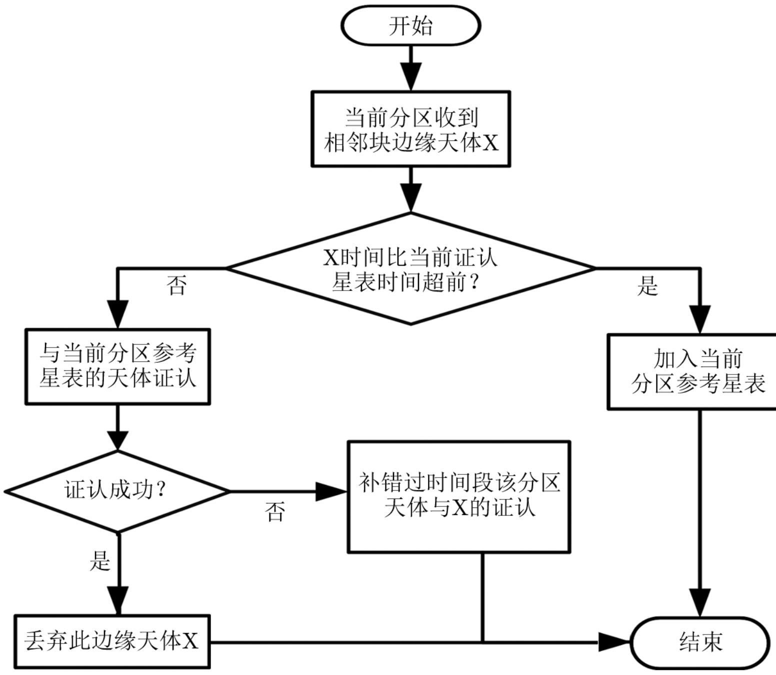


图 1　参考星表边缘天体跨节点更新流程图

(1) 相邻块证认进度超前。此时需要考虑一种可能，若边缘天体 X 是个超新星，它的首次出现有可能就在此相邻块，在将 X 更新到相邻块参考星表之前它已经存在。如图 2，A 块 15 号出现一个新天体 X，因为 X 处于 A、B 间的边缘位置，将其加入 A 块参考星表的同时发送给 B 块所在节点。若 X 首次出现在 B 块中，发现时间为 10 号且已经将其发送给 A，则 A 块就不会在 15 号出现新天体 X。因为发送有时延，没有及时更新到 A 块的参考星所以才会再次被发现，而 B 块在 15 号之前已经出现天体 X。因此，当 B 块所在节点收到天体 X 时，只需与 B 块参考星表 11 号至 25 号的新增天体数据进行证认：若证认成功了，丢弃此 X，因为 X 在B 中之前已被发现；若证认失败，加入 B 块的参考星表中并与 B 块 11 号至 25 号这段时间所有天体进行证认。

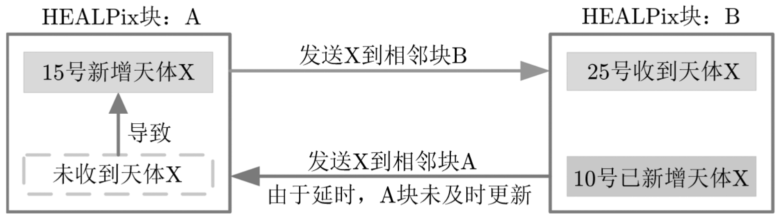


图 2　相邻块证认进度超前边缘更新示意图

(2) 相邻块证认进度落后。将边缘天体加入相邻块的参考星表中，根据天体拍摄时间和后续天体进行证认，若后续星表中的天体拍摄时间比该天体拍摄时间早，则不进行证认。

3　两表间不同 Join 策略对比

为了实现相同块号下两两星体的距离计算和阈值判断，本文考虑了两种方法：一种方法是利用数据库中的连接 (Join) 操作；另一种则是不利用 Join 操作，而直接采用 Spark 中 HDFS 文件数据和算子操作。本章详细介绍基于 Join 的方法，非 Join 方法详见 4.2 节。

为了实现距离计算和阈值过滤，采用不等值连接，连接条件即为参考星表中的一条记录与待证认表中一条记录的球面距离值小于阈值，即公式 (1)－(5)，连接成功的结果即为证认成功的记录对。为了提高处理的性能，采用哈希连接 (Hash Join)，将连接字段的值映射到哈希表中的桶中，以减少 reshuffle 或重排序，对于大规模数据有良好的适应性。

考虑到参考星表的数据需要和连续时间轴上的其他星表进行一一证认计算，所以将参考星表数据作为小表，其他连续时间轴上的星表整合后作为大表，为了进一步提高性能，实验对比了两种适合于小表和大表间的哈希连接策略：Shuffle Hash Join (SHJ) 和 Broadcast Hash Join (BHJ)。

SHJ 的实现是将大表和小表根据连接的键按照 HashPartitioner 方法进行重新分区，即 Shuffle 操作。目的是让两张表中相同键的数据分在一个分区内，然后对同一个分区内的数据执行连接操作。

BHJ 的实现是将小表的数据广播到 Spark 的其他节点，然后在每个节点上独立执行连接操作，最大限度地利用 CPU 资源专注于执行连接操作。这种 Join 策略在一定程度上牺牲了节点空间来换取 Shuffle 操作可能带来的耗时操作，但广播的表如果较大则可能对驱动器节点造成较大的压力，因为广播大表的代价是昂贵的。

本文将基于这两种 Join 策略分别实现两表间的证认计算，并展开对比研究。实验结果见 5.3 节，实验表明：当计算资源足够时两表间采用 Join 操作进行证认计算，选择 BHJ 更有优势；而当计算资源不足时应选择 SHJ。

图 3 给出了两表间 Join 流程图，首先用 textFile 算子读取 HDFS 元数据创建为 RDD (Resilient Distributed Datasets)[13]，然后用 map 算子和 toDF 函数将 RDD 转换为

DataFrame。根据 Healpix 索引将两表连接返回新的 DataFrame，再使用 withColumn 函数新增一列 (两个天体之间的角距离) 并使用 filter 函数过滤出角距离小于阈值的数据，最后将符合条件的数据输出至 HDFS (Hadoop Distributed File System)[14]，完成星表数据的时序重组过程。

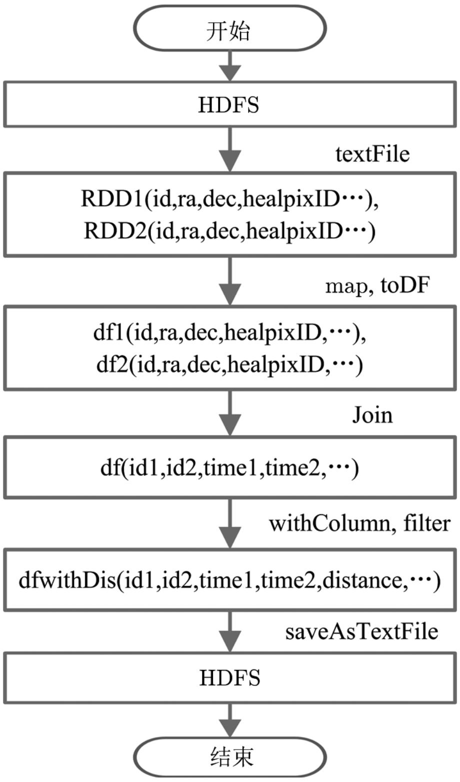


图 3　两表间 Join 流程图

4　基于 Spark 的海量星表数据时序重构方法和重点效率突破

4.1　基于 Spark 的海量星表数据时序重构方法设计架构

本文提出一种基于 Spark 的异步非阻塞模式下的海量星表时序重构方法，其整体框架如图 4 所示。预处理阶段，有效地对原始星表数据进行过滤，从而降低存储和证认成本并生成元数据文件，然后计算 HEALPix 索引并将数据进行格式化处理。值得注意的是，天体所在块的相邻块索引也须记录下来，以便在参考星表中冗余存放相邻块数据的边缘数据。最后通过 Spark 自定义分区函数将星表数据划分在不同的 HEALPix 区块文件中，等待下一步证认计算。接下来将预处理后的星表进行分布式证认计算，最后阶段生成天体的时序数据。

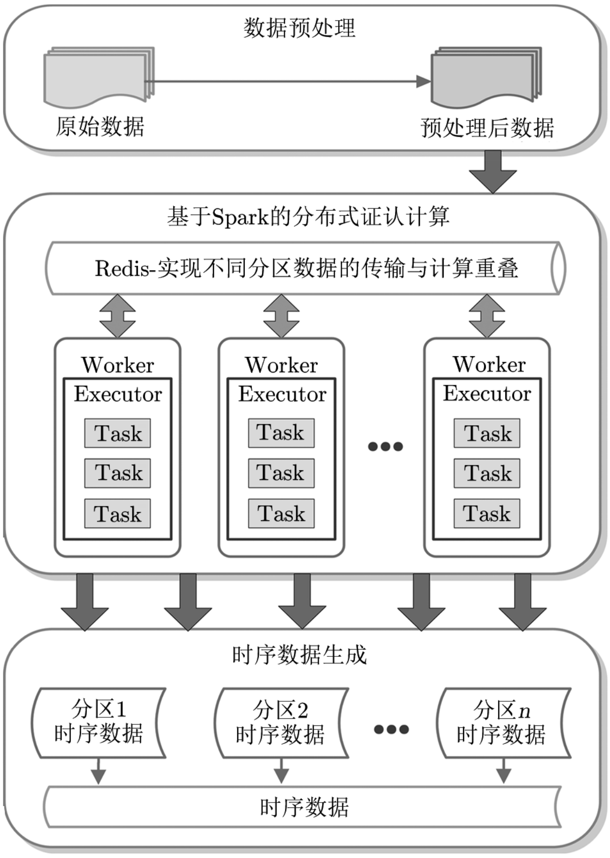


图 4　基于 Spark 的海量星表数据时序重构方法整体框架图

4.2　免 Join 证认计算方法

第 3 章介绍了将距离计算和阈值过滤集成于数据库连接操作的方法，这种方法编程简单，实现方便，但频繁的 Join 操作仍会对效率产生一定影响。为了进一步优化效率，这里考虑了一种直接将距离计算公式和阈值过滤作用于 HDFS 数据文件的免 Join 方法。不同 HEALPix 分区数据并行执行证认计算任务，需要频繁读写的参考星表存储在内存中，连续时间轴上星表按照时间顺序依次和参考星表中的天体进行证认计算，证认计算流程如图 5 所示。这种方法将与本文基于 Join 操作的方法，以及前人优秀方法进行对比。

(1) 从 HDFS 中读取星表数据后首先创建当前分区的参考星表。

(2) 若当前分区有待证认天体且 Redis 数据库中有当前分区边缘天体，则依照参考星表更新同步问题的解决方案，将边缘天体更新到参考星表中。

(3) 若 Redis 数据库中没有当前分区边缘天体或者已经更新完成，则将下一时间段星表和参考星表进行证认计算：若匹配失败则认为该天体为新天体，将其加入参考星表中；若匹配成功则将两个天体数据进行记录。

(4) 如果新天体为边缘天体，则说明相邻分区可能发生了边缘漏源，首先将其加入

Reids 数据中，通过步骤 (2) 该边缘天体会被加入相邻分区的参考星表中，通过参考星表中冗余存放边缘天体，避免边缘漏源的发生。

(5) 重复步骤 (2) 直到当前分区全部时间段的星表完成证认。

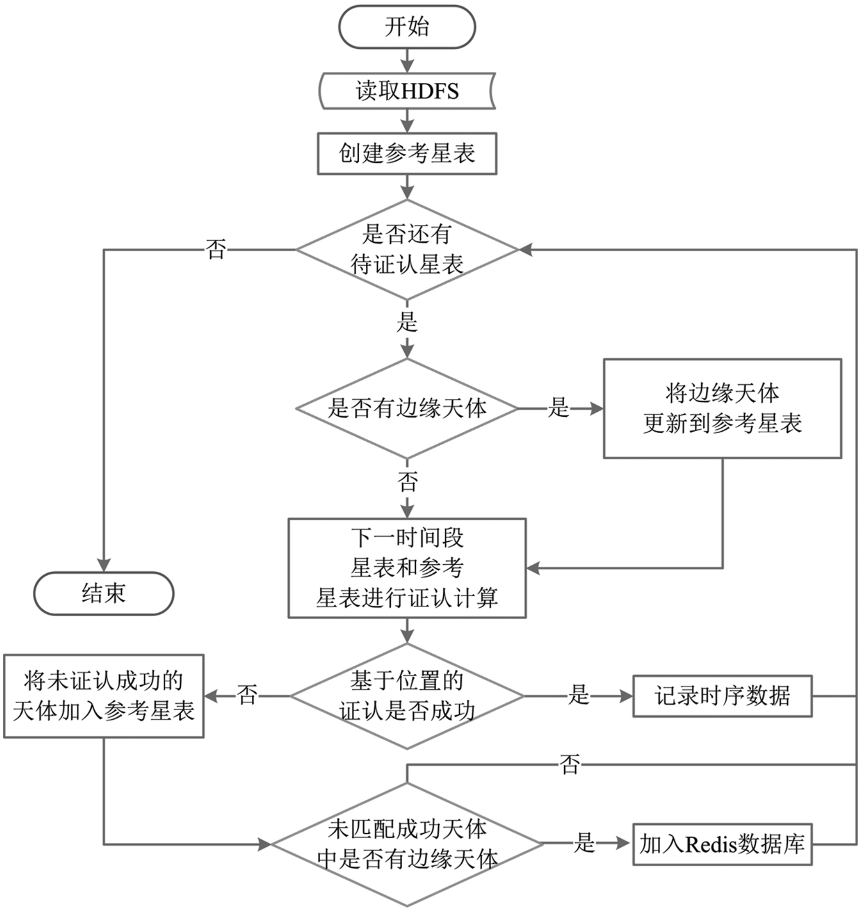


图 5　基于 Spark 的时序重构计算流程图

5　实验分析

5.1　实验环境与数据集

实验使用 Spark 分布式集群作为计算平台，Spark 集群采用 3 台阿里云服务器进行搭建，其中 Master 节点配置在 1 台服务器，Worker 节点配置在 3 台服务器，每台服务器使用相同的配置，具体配置参数如表 1 所示。Spark 采用 Yarn-cluster 运行模式，每个计算节点分配 2 核逻辑 CPU 和 8 GB 的内存。不同实验将虚拟化不同的计算节点进行计算和分析。

本文利用南极巡天望远镜 AST3-I 和 AST3-II 分别在 2012 年和 2016 年观测的真实星表

表 1　环境配置参数

参数	值
CPU 型号	Intel(R) Xeon(R) Platinum 8369B
逻辑 CPU 数量	64
内存大小	64 GB
操作系统	CentOS Linux release 7.8.2003
JDK 版本	JDK 1.8.0_261
Scala 版本	2.12.12
Redis 版本	5.0.14.1
Hadoop 版本	3.1.2
Spark 版本	3.0.0

数据进行实验。如表 2 所示，数据集信息包含了数据来源、数据集名称、文件数目、原始文件大小和预处理后文件大小。实验源代码已上传至 Github (https://github.com/Liltsliverfish/A-time-series-reconstruction-method-of-massive-astronomical-catalogues-based-on-Spark)。

表 2　数据集信息

数据集来源	数据集名称	文件数目	原文件大小/GB	预处理后文件大小/GB
AST3-II	Tess4	1 882	2.23	0.53
AST3-I	HD88500	591	2.29	1.89
AST3-I	HD117688	655	6.36	5.31
AST3-I	HD136488	16 874	8.3	5.50
AST3-I	HD14341	660	11.3	6.90
AST3-II	Tess5-13	1 442	19.7	9.30
AST3-I	Transit	3 158	93.82	78.21

5.2　星表预处理性能评价

为了分析星表文件的预处理性能，图 6 给出了不同数据集预处理前后的文件大小和预处理时间。AST3-I 的数据集预处理后星表文件大约是原始文件的 28%，AST3-II 的数据集预处理后约为原始文件的 83%。预处理时间在可以接受的范围内。根据处理后的文件大小可知，预处理原始星表数据可以极大地降低数据的存储量，节省存储时间，同时也节省不必要的跨节点数据传输时间，从而提高时序重构的效率。

5.3　两表间不同 Join 算法对比

为了验证 BHJ 和 SHJ 两种 Join 算法下两表间证认计算的表现，实验中利用 Tess4 数据集在不同节点和不同数据量下分别进行了 Join 计算。

由图 7a) 可知，当星表数据量一定时，不同节点下两表间采用 BHJ 算法证认计算的运行时间都少于 SHJ。当计算节点数为 16 时，采用 BHJ 证认计算的用时相比采用 SHJ 减少 56.6%。由图 7b) 可知，当计算节点为 32 时，在不同星表数据量的情况下，采用 BHJ 计算的用时同样少于采用 SHJ。因为 SHJ 算法的 Shuffle 操作十分耗时，其会导致星表数据在不

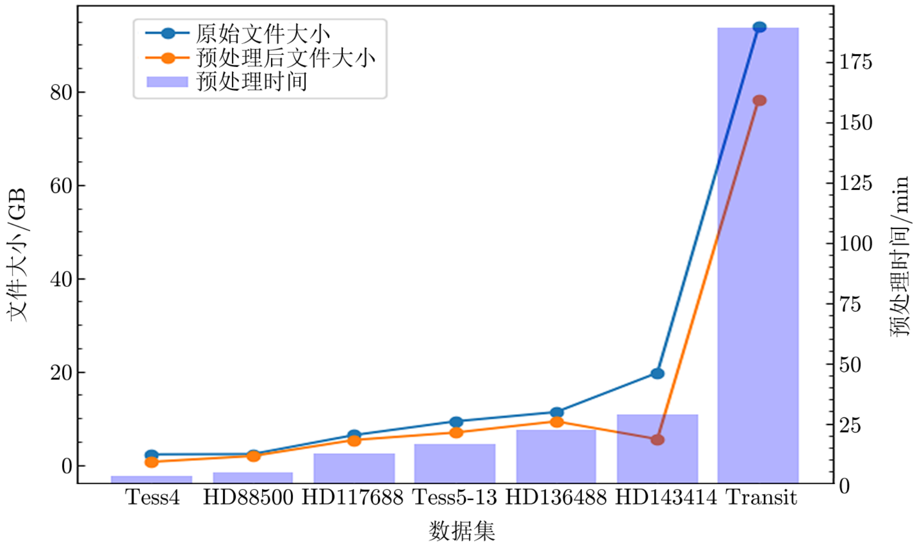
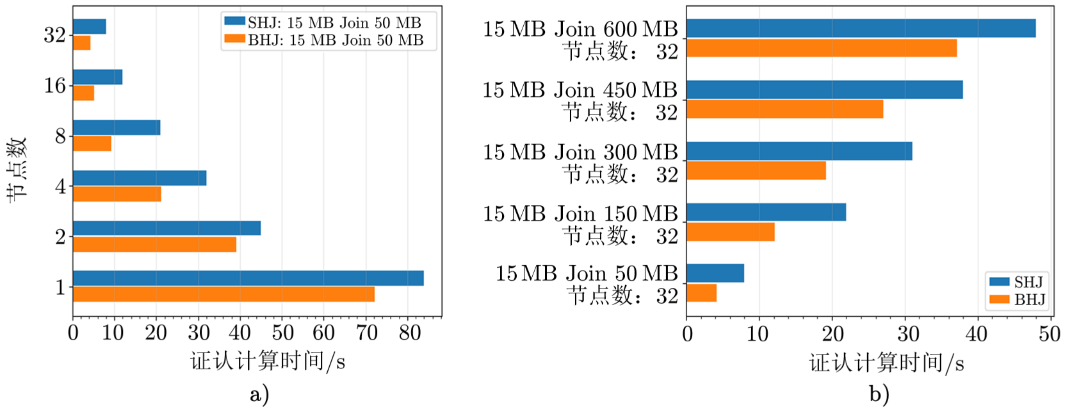


图 6　**预处理性能** (HEALPix **层级**=10)

同节点间进行传输。值得注意的是，当实验中配置计算节点内存为 1 GB，两表数据大小分别达到 150 MB 和 600 MB 时，采用 BHJ 算法出现了内存不足的情况，进而导致证认计算失败。因此，当计算资源足够时，两表间进行 Join 操作进行证认计算选择 BHJ 更有优势；而当计算资源不足时应选择 SHJ。



注：a) 不同计算节点数两表间 Join 对比；b) 不同数据量两表间 Join 对比。

图 7　**两表间 Join 对比**

5.4　免 Join 证认方法性能评价

5.4.1　不同 HEALPix 层级免 Join 证认方法表现

不同 HEALPix 层级对星表划分粒度不同，层级越深划分粒度越细，即分块越多，边缘数据也随之增多，因此需要额外的时间来处理边缘数据，同时也涉及边缘数据的跨节点更

新。层级数划分越小，则每个分区包含的天体数量增多，计算量也会随之增大。对于同源星表，大多数情况下目标天区的天体分布不均匀。在本文实验中的星表划分中，一些分区的天体数量甚至只有个位数，所以当层级划分得太小时，天体数量少的节点大多数时间处于空闲状态，从而浪费了计算资源，无法充分发挥分布计算的性能。因此，HEALPix 层级选取对于时序重构的性能有极大的影响，需要提前选择最优的 HEALPix 层级，降低计算的复杂度。

为了选取合适的 HEALpix 层级，实验中利用 64 个计算节点在不同 HEALPix 层级下分别对多个数据量相近的数据集进行了时序重构的实验。如图 8 所示，对于实验中的所有数据集，HEALPix 层级为 10 的计算运行时间最少，降至 8 和 9 或升至 11 和 12，计算时间都会增加。由于 HEALPix 是四叉树结构，HEALPix 层级每升一级，星表的分块数量增加约 4 倍，导致证认计算的任务数量增加。如果任务太多，任务启动和切换开销便会大量增加，导致性能下降。HEALPix 层级每降一级，星表的分块数量减至 1/4，但是每个分区包含的数据量增加为原来的 4 倍左右，因而导致任务数量太少，无法充分发挥出集群的并行计算能力，任务执行过慢，同时还有可能会出现内存溢出 (out of memory) 的异常。对于本文的实验环境，HEALPix 划分的层级为 10 级时是最优的选择，在后续实验中 HEALPix 层级都为 10 级。当然，不同的测试环境中，此值可能会存在差异。

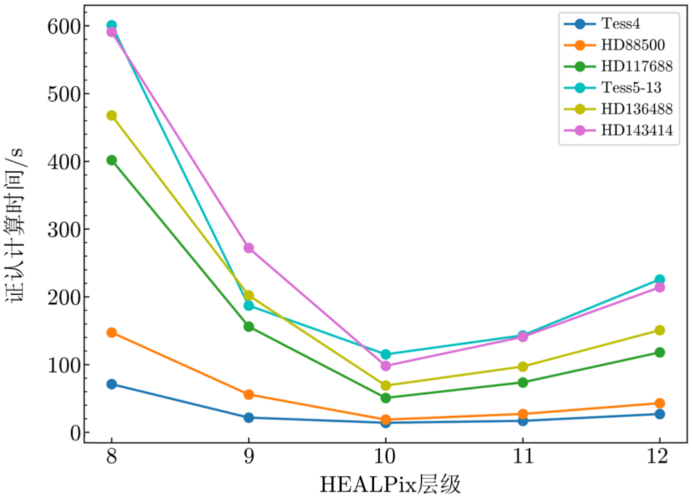


图 8 不同 HEALPix 层级下免 Join 证认计算时间

5.4.2 不同节点免 Join 方法性能评价

为了满足时序重构中的效率和精度要求，本文提出了基于 Spark 的异步非阻塞模式的免 Join 证认方法，该方法突破传统的 Join 算法，可以进一步提高时序重构的效率。

由图 9 可知，每个数据集的证认计算时间随着计算节点数的增加而减少。当计算节点数一定时，除 Tess5-13 数据集之外，证认计算时间随着数据集文件大小的增加而增加。Tess5-13 数据集的数据文件总大小与 HD117688 数据集比较接近，但是 Tess5-13 数据集证

认计算用时比 HD117688 数据集多了约 56%，这是由于 Tess5-13 数据集的边缘数据较多，需要花费时间来处理边缘数据，因此 Tess5-13 数据集的计算用时多。

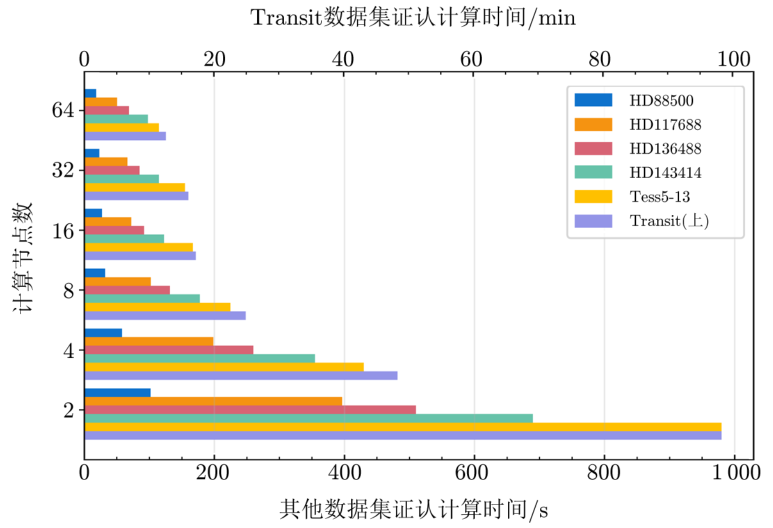


图 9　不同 HEALPix 层级下免 Join 证认计算时间

5.4.3　免 Join 证认方法综合性能评价

为了验证免 Join 方法的整体性能，实验中利用 AST3-I 数据集在不同计算节点下进行了实验，其中数据集大小为 100 GB。如图 10 所示，证认计算时间随着节点数的增加而减少，在计算节点为 64 时，证认计算时间仅用为 6.18 min。

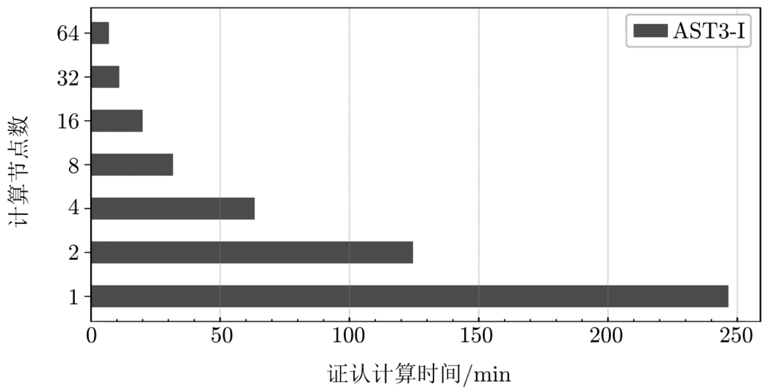


图 10　不同节点下综合实验性能表现

图 11 给出不同计算节点下的加速比，随着计算节点的增加，加速的效果开始显现，当然其中一部分是因为参考星表的副本策略和降低跨节点数据传输量的出色表现。随着计算节点的增加，证认计算消耗的时间逐渐减少，基本呈线性增长趋势。

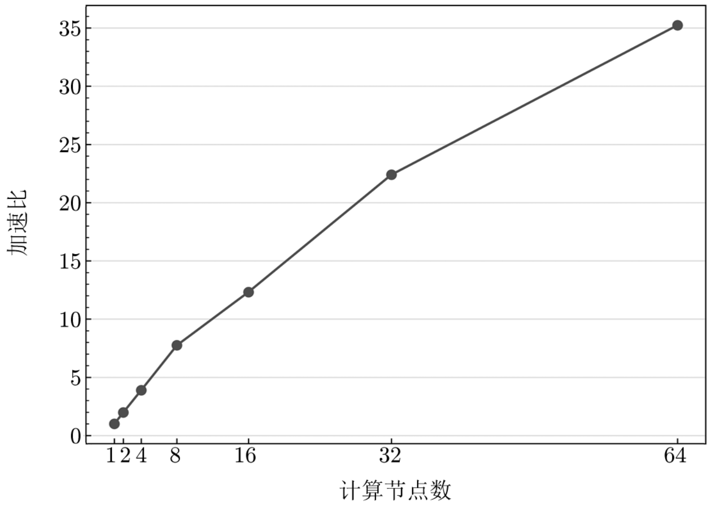


图 11　综合实验加速比随计算节点数的变化

5.4.4　与他人工作相比

当前，连续多星表间的时序重构工作还较为少见。Yu 等人[7]的研究工作与本文工作最为接近。他们通过 MPI 编程实现并行计算，实验环境的操作系统为 Ubuntu、处理器为 Intel i7-4790 CPU (8 cores @ 3.6 GHz)、内存大小为 16 GB。本文使用相同的数据集与他们在 6 个进程下的实验结果进行比较。结果显示，虽然 Yu 等人方法提供了可视化界面，在使用性方面更具有优势，但是证认计算方面利用本文方法采用 5 个计算节点可以超越其性能。从图 12 可知，当计算节点数继续增加，本文方法可以获得持续性性能增加。当节点为 64 个时，HD88500、HD117688 和 HD136488 数据集证认计算的耗时分别是 Yu 等人方法耗时的 30%、18% 和 19%。可以看出，本文算法在性能上有了较大幅度提升。而且，随着分布式环境中节点数量的增加，本文的方法可以达到接近线性的加速比，这是因为方法中只对参考星表进行冗余存储，其他星表按照 HEALPix 特定级别进行分块处理。可见，主要的并行方式是数据并行，不同节点间的计算任务彼此独立，且新增星体的多节点同步更新和追加证认采用异步形式，因此证认计算过程中，很少涉及跨界点数据传输，并且随计算节点的增加，算法并行执行效率可以基本达到线性加速比。综上所述，本文基于 Spark 的大规模分布式架构和良好的加速比使本方法具有良好的可扩展性，对于数据量不断增加的时域天文观测设备来说具有良好的适应性。此外，本文时序重构方法只需要一种索引方式，有效节省了数据存储空间。在计算精度方面，本文方法的理论漏源率是 0，Yu 等人采用 HTM 索引和 HEALPix 索引混合计算的漏源率约为 4.5%。可见，引入新的并行框架、进行算法设计对程序性能的提升起到决定性的作用。

5.4.5　不同证认方法的比较

为了对比免 Join 证认方法与两种 Join 算法的性能，本文利用 HD88500 数据集在 64 个计算节点下进行了实验。从表 3 的实验结果可以看出，免 Join 证认方法显然比传统基于

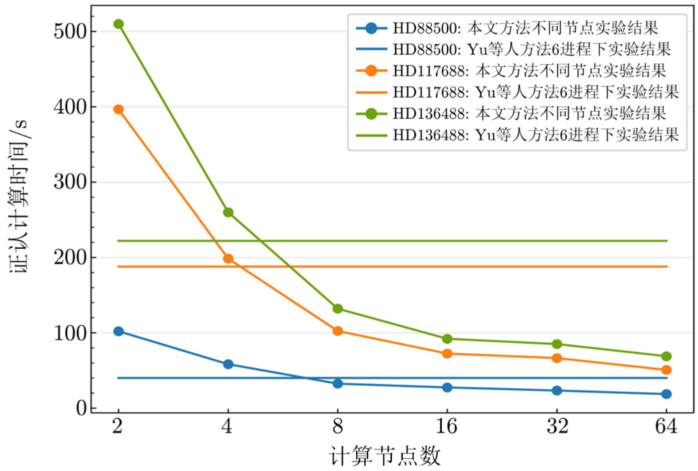


图 12　与他人方法比较

Join 的证认方法更加高效。基于 Join 的证认计算方法存在跨节点数据传输以及数据倾斜的问题，而免 Join 证认计算方法通过前期的预处理避免了大量的跨节点数据传输，大大减少了证认计算的时间，提高了证认的效率，可以更快得出证认计算结果。

表 3　不同证认方法比较

证认方法	SHJ	BHJ	免 Join
证认计算时间/s	119.5	87.9	12

6　结　语

本文针对海量天文星表数据的时序重构面临的挑战，结合星表索引、算法设计、数据更新等问题的最新理论方法，解决了海量天文星表时序重构的效率和精度问题，为后续的研究提供了数据支持。本文首先提出异步非阻塞模式下的时序重构方法，提出基于参考星表副本策略，该策略在保证精度的前提下减少了数据传输量。针对分布式环境下的数据跨节点更新和同步问题，本文对于不同的情况给出了相应的解决方法。此外本文还对两表间采用 Broadcast Hash Join 和 Shuffle Hash Join 两种 Join 算法从理论和实验两个角度进行了研究和对比分析，分别给出了适用的情况。最后基于 Spark 分布式框架提出免 Join 的时序重构方法，更加高效地实现了海量天文星表的时序重构。实验表明，本文设计的时序重构算法是高效的、可行的、有实用价值的，对于时域天文学的发展有一定促进作用；基于 Spark 的大规模分布式计算架构和接近线性的加速比，使本方法对于 GWAC 等更大规模的时域望远镜的星表时序重构任务同样具有参考价值。

参考文献：

[1] Cordier B, Wei J, Atteia J L, et al. Proceddings of Science, DOI: https://doi.org/10.22323/1.233.0005, 2015
[2] Vallenari A, Brown A G A, Prusti T, et al. A&A, 2023, 674: A1(2023)
[3] 万萌, 吴潮, Ying Zhang, 等. 天文研究与技术, 2016, 13(3): 373
[4] Ivezic Z, Tyson J A, Acosta E, et al. AJ, 2019, 873(2): 111
[5] Zhao Q, Sun J, Yu C, et al. Transactions of Tianjin University, 2011, 17(1): 62
[6] Du P, Ren J J, Pan J C, et al. Science China: Physics, Mechanics and Astronomy, 2014, 57(3): 577
[7] Yu C, Li K, Tang S, et al. MNRAS, 2020, 496(1): 629
[8] 徐丹滢, 赵青, 权文利, 等. 天文学进展, 2022, 40(2): 298
[9] Brahem M, Yeh L, Zeitouni K. Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Seattle, WA: ACM, 2018: 229
[10] Zečević P, Slater C T, Jurić M, et al. AJ, 2019, 158(1): 37
[11] 刘莹, 马智, 游子毅, 等. 天文学报. 2022, 63(3): 10
[12] Armbrust M, Xin R S, Lian C, et al. Proceedings of the 2015 ACM SIGMOD international conference on management of data, New York: ACM, 2015: 1383
[13] Zaharia M, Chowdhury M, Das T, et al. Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, Berkeley: USENIX Association, 2012: 15
[14] Shvachko K, Kuang H, Radia S, et al. Proceedings of the IEEE 26th symposium on mass storage systems and technologies (MSST), USA: IEEE, 2010: 10

Research on Time Series Reconstruction Method of Massive Astronomical Catalogues Based on Spark Distributed Framework

ZHAO Qing[1], QUAN Wen-li[1],
CHEN Ya-rui[1,2], CUI Chen-zhou[3], FAN Dong-wei[3]

(1. College of Artificial Intelligence, Tianjin University of Science and Technology, Tianjin 300457, China; 2. Engineering Research Center for Integration and Application of E-Learning Technology, Ministry of Education, Beijing 100039, China; 3. National Astronomical Observatories, Chinese Academy of Science, Beijing 100101, China)

Abstract: Time series reconstruction is a crucial data processing step in time domain astronomy and serves as the foundation for fitting light curves and conducting time domain analysis. For many large-field time domain surveys, it is necessary to complete this computational process within a single exposure cycle. With the rapid increase in astronomical data, existing methods for astronomical data processing struggle to simultaneously meet the accuracy and efficiency requirements of time-series reconstruction. The memory-based computing general-purpose distributed framework, Spark, holds the potential to improve the

efficiency of this process. However, applying Spark directly often encounters issues. MapReduce distributed models like Hadoop and Spark require relatively independent tasks among distributed cluster nodes and minimal data transfer across nodes during execution. Otherwise, frequent communication becomes an efficiency bottleneck for the application of the model. However, due to the presence of boundary problems in cross-matching, it is inevitable to transmit newly added data at the boundaries, severely restricting the concurrency of the model and reducing the acceleration ratio in practical parallel model applications. Therefore, we propose a non-blocking asynchronous execution flow, where each distributed process handles continuous processing exclusively for independent sky regions. The delayed batch appending of additional identification tasks from block-edge newly added celestial bodies in other nodes is determined based on the progress of each process. This ensures that identification calculations are not omitted, thereby improving concurrent efficiency while maintaining algorithm accuracy. Additionally, a research study was conducted on different join strategies between two tables, examining them from both theoretical and experimental perspectives. Furthermore, a join-free strategy was proposed. Finally, the design of an efficient time-series reconstruction system based on the Spark distributed framework validates the aforementioned research. Experimental results demonstrate a significant improvement in the efficiency of the proposed time-series reconstruction algorithm compared to previous research, laying a solid foundation for the analysis of astronomical time-series data in time-domain astronomy.

Key words: time domain astronomy; cross-match calculation; time series reconstruction; distributed computation; Spark